Bachelor's thesis

Information Technology

2015

Artur Oleynik

ANDROID KIOSK LAUNCHER

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR´S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

June 2015| 34

Instructor: Patric Granholm

Artur Oleynik

ANDROID KIOSK LAUNHER

This Bachelor's thesis is following development of a kiosk launcher for Android operating system. The purpose of this thesis was to create an Android program that would answer the requirements of the commissioning company.

The commissioning company, Pointer Group Oy, is a small local business in Turku that provides feedback-gathering services in Finland for other companies. The Launcher is used in a questionnaire device in public places such as banks and restaurants to collect user feedback on the products and services offered by the companies that ordered the device. The application provides the environment for the questionnaire website to run on the device and restricts access to general and network settings for the user. At the same time, the authorized users have the freedom to modify the configuration of the device.

Tools that were used in the project are Android SDK and the Java programming language, the integrated development environment Eclipse and Android studio.

The resulting program was approved by the company and used in their business although some of the requirements had to be dropped due to time limitations and technical reasons, for instance, the audio and video playback. The requirements that had to be dropped due to the time limit are scheduled to be added in the future updates.

KEYWORDS:

IT, Android, launcher, application development.

CONTENTS

PICTURES

FIGURES

TABLES

LIST OF ABBREVIATIONS (OR) SYMBOLS

Google Play	Originally the Android Market, is a digital distribution platform operated by Google.
HTML	HTML, which stands for HyperText Markup Language, is the predominant markup language for web pages.
OS	Operating System
CSS	Cascading Style Sheets (CSS) is a style sheet language used for describing the look and formatting of a document written in a markup language.
SDK	Software Development Kit
IDE	An integrated development environment (IDE) or interactive development environment is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger.

1 INTRODUCTION

The aim of this thesis is to develop an Android Kiosk Launcher. This thesis will explain the requirements and features for this type of program as well as problems one might encounter during coding and their solutions.

Android is a fast-growing operating system for smart phones, tablets, TVs and embedded devices unveiled by Google in 2007 and released under open source licenses in 2008. Because of its open source nature, Android has vast opportunities for customization by large manufacturers, such as Samsung, HTC, Sony as well as smaller companies like Oppo, Xiaomi, and Huawei.

Nowadays Google's operating system is a leading one with over 80% in smart phone market share:

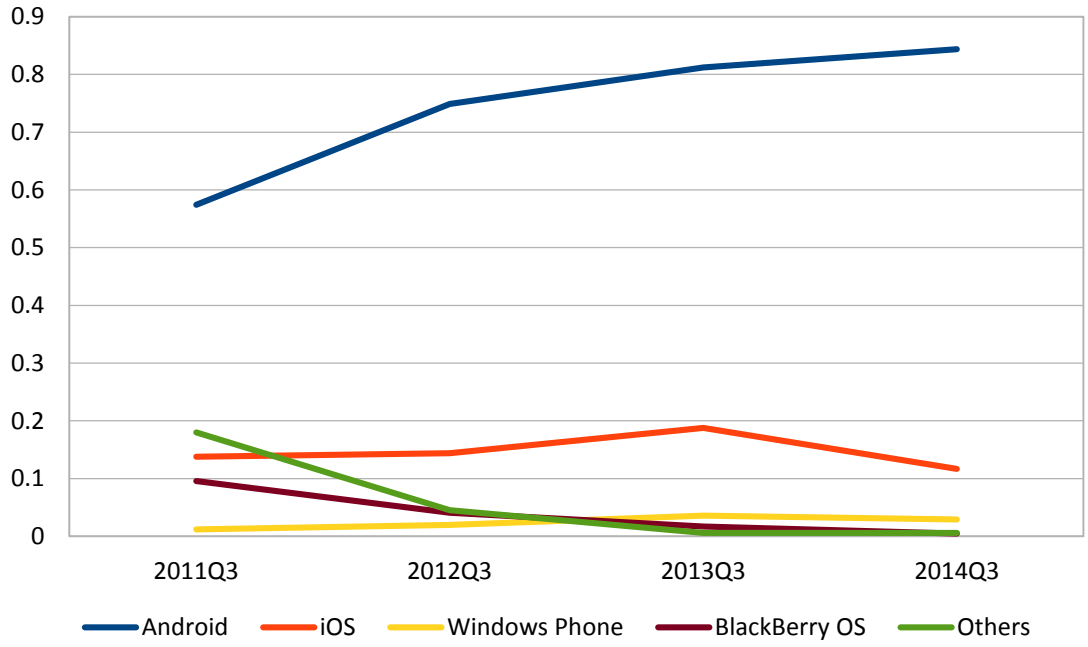


Figure 1. Worldwide Smartphone OS Market Share (IDC, 2014 Q3)

This thesis project will be focused on creating a Kiosk Launcher for tablet use as primary and Android is a dominant platform on tablets, which can be seen from Figure 2.

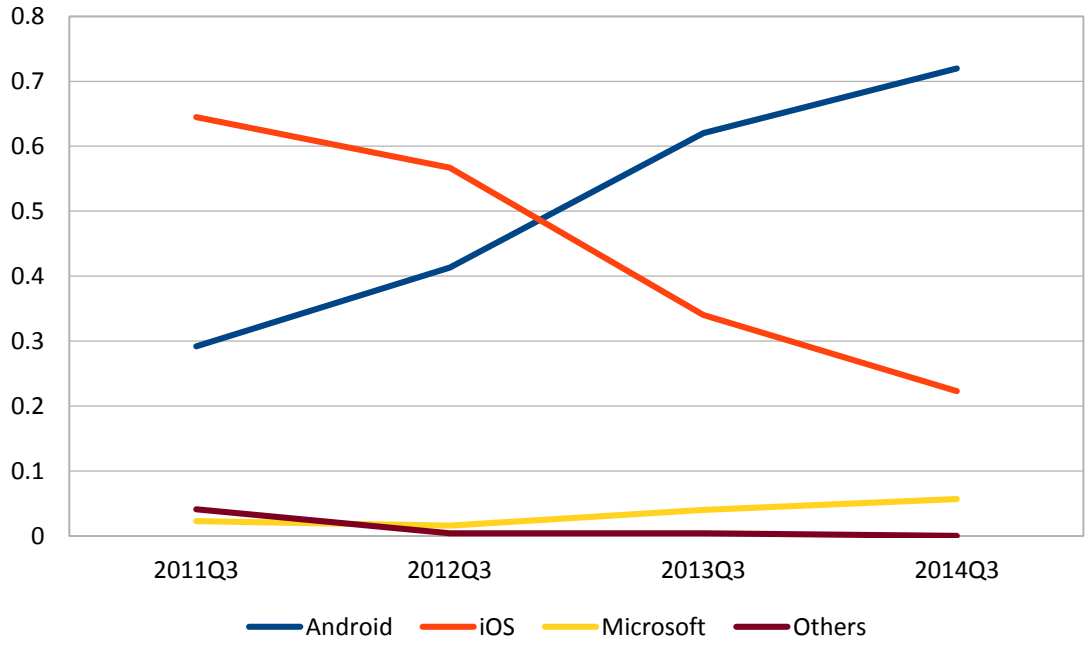


Figure 2. Worldwide Tablet OS Market Share (Statista, 2015)

> "What is an Android launcher? Probably the most powerful feature of Android is its ability to be customized. And that starts with what's typically called the "launcher." The launcher usually is considered to be the homescreens and app drawer, and they come in all sorts of flavors and designs."(Phil Nickinson 2012)

The aim of the project was to develop Android Kiosk Launcher that would meet the requirements of the commissioning company. The commissioning company, Pointer Group Oy is a small local business in Turku that provides feedback-gathering services in Finland for other companies. The Launcher is used in a questionnaire device in public places such as banks and restaurants to collect user feedback on the products and services offered by the companies that ordered the device.

2 REQUIREMENTS

This section states the requirements set to the launcher by the company.

2.1 Application must-have requirements

Kiosk Launcher should feel native on the device, work fluently and flawlessly and have following features to be finely tuned for the use by the commissioning company

- Identify itself as a launcher

Android manifest should state the program to be a launcher.

- Be able to replace the default launcher

This should be automatic if 2.1.1 is set up correctly.

- Run on full screen without any system bars.

This is currently performed by a third-party application, but should be done by the app itself.

- Have webview with multiple configurations

Different webviews specified later in Section 2.2.

- Have hidden settings access with at least two access levels

The hidden menu button that with long press (length defined in preferences) should give a small prompt for the PIN code (set in preferences). Preferably two different PIN codes and access levels should be defined, one for Wi-Fi settings only and other for full system settings. The PIN prompt should timeout and

disappear within few seconds if not filled. Wrong PIN code attempts should be logged in the system protected file.

- Support HTML5, CSS3, Javascript, and possibly video and audio playback

The current version of the product does not need anything except HTML5, CSS3, and JavaScript. Video and audio support might be needed for future products.

- Support offline usage, i.e., store content locally until a connection is available

The device must be able to handle at least necessary file storage usage while being offline. The current setup handles offline usage as long as it is on network when it started up. If startup at offline stage is supported, it should be defined in settings.

- Have all settings in clear human readable xml file

All configurable items in application should be defined in `/data/data/<apk>/shared_prefs/<somename>.xml` or similar. The application must support silent install (adb install <application>) and startup by activity manager call such as `am start com.example.<application>/.MainActivity`.

- Show battery status

The battery status should be displayed by icons of several levels (preferably five) in the bottom left corner. Whenever the device is charging, there should be an icon that represents that.

- Show Wi-Fi status

Wi-Fi status should also be displayed by another icon with different levels near the battery status icon. Whenever there is a connection to the server, this should be represented on the Wi-Fi status.

2.2 Webviews

All webviews below also need to support a vertical orientation mode!

- Fullscreen Webview without any browser controls (current product usage)

A fullscreen webview without any visible buttons or bars should be displayed. This is the default mode for the current generation operation, fullpage (1024x768 or 2041x1536) webview with HTML5, CSS3 and JavaScript support.

This view can be also used for future product video-only mode or a separate app might be implemented for that functionality. If incorporated with this one, then we also have requirements for supporting Arduino USB devices and listening/sending events from/to these devices. Touch needs to be possibly blocked in this case then as well.

- Fullscreen Webview with browser controls (back, forward, home)

This is the future product's "Showroom" mode where our software is running a predefined customer web page and nothing more. There should be no address bar visible, only home and back and forward buttons. Some kind of "loading" indicator needs to be implemented for this mode.

Pressing the "home" onscreen button should also clean all session data, i.e., if the user used possible webshop and paid with his credit card, all that information should be deleted permanently from the device when "home" pressed. This mode should also have "inactivity timeout" in settings after which the browser should reset all data and return home.

- Fullscreen Webview with tabs and browser controls (back, forward, home)

This requirement is identical to previous one except that this should have several tabs with pre-selected content and one tab set as default.

2.3 Tools

Tools that were used in the project include but not limited to:

- Eclipse
- Android studio
- Android Development Tools(ADT)
- Android SDK

2.4 Device requirements

Following requirements describe Android tablets that are ideal for the commissioning company use and those devices are supported by the Kiosk Launcher:

- Android 4.1 or higher
- 9.7 inches screen size or higher
- Boot on charger plug in
- Root installed

3 SIMILAR PROGRAMS

There are several different Kiosk Launchers for Android already on the Play Market and to be able to create the launcher that is best suited for the intended use, one should study the competitors at their best.

In this chapter, several popular alternatives are examined and their advantages and disadvantages if they were used in the project. The commissioning company that ordered the application tried all of those launchers for their business and came to conclusion that they needed a new software program that is built from scratch and finely tuned for specific usage since all of the competitor programs lack some important features.

Kiosk Browser Lockdown

By: ProCo IT

Downloads: 10 000

Rating: 4.2

Description:

Picture 1. Launcher Logo 1

Kiosk Browser developers claim that their application has been designed for use on any Android device, whether tablet, phone or else and that its primary usage is in creating public kiosks, interactive digital signage, etc. The developers also state that their launcher locks down the user interface for security purposes so that the end-user is not able to modify Android system configuration or gain access to not allowed third-party software.(ProCo IT, 2015)

Advantages:

- Can be set as default launcher
- Has access to settings requires password
- Default web page is configurable
- Whitelist websites are used to allow exclusive access to them

- Screen can be set on permanently
- Rotation can be locked
- System back/home buttons can be locked
- Show/Hide system dialogs
- Recent apps screen can be disabled
- Fullscreen mode (removes action bar, bottom bar on 4.4 devices and bottom bar for 4.0 - 4.3 where device is rooted).
- Several ways to exit full screen mode: swipe from top edge of screen, press volume button or back button
- Video/pictures/website screensaver (pro version)
- XML Import/Export for mass device configuration (pro version)
- RMC (remote management console) subscription available

Disadvantages:

- No way to hide navigation bar permanently, therefore access to system settings is not restricted
- No support for offline usage, i.e., stores content locally until connection available
- No access to settings from within the launcher for administrative use.

SureLock Kiosk Lockdown

Developed by: 42Gears Mobility Systems

Downloads: 50 000

Rating: 3.9

Description:

Picture 2. Launcher Logo 2

SureLock is an application for Android based tablets and smartphones that can lock it down in order to prevent unwanted device abuse. It gives the ability to configure which software and device features can be used by the end-users. In most cases, users may be allowed access to just one or two apps and SureLock

will ensure that the users cannot access anything other than those allowed applications (42Gears Mobility Systems, 2015)

This program is useful in cases when the Kiosk Launcher in place cannot restrict access to system settings or third-party applications. It is a powerful tool, but it has to be used in conjunction with something else since it does not have web browser functionality.

Advantages:

- Locks down Android smartphones and tablets into kiosk mode
- Restricts users to allowed applications only
- Hides home button and disable bottom bar
- Blocks users from changing System Settings
- Admin can securely unlock the device with a password
- Password control
- Auto launch application(s) at startup
- Lockdown of peripherals (wifi, bluetooth, auto-orientation, flight-mode, audio)
- Remote Management Console subscription available
- Imports/Exports settings to xml
- Deploys SureLock remotely (http or file transfer)

Disadvantages:

- Website access done through separate branded application SureFox
- Paid version is expensive per installation (around €50)
- Free version lacks features and has interruptions for advertisements
- Small list of supported devices

Kiosk Browser

Picture 3. Launcher Logo 3

Developed by: Andy Powell

Downloads: 10 000

Rating: 3.6

Description:

Kiosk Browser is a browser than can run in normal or full screen mode and allows restricting to a specific site if required. Kiosk Browser can also be configured to allow external sites and also to force any external sites to the default system browser.(Andy Powell, 2015)

With built-in javascript commands and fixed or variable zoom, it can present and control the browsing experience for its users. Kiosk Browser supports Flash and other browser plugins.

Kiosk Browser allows running other applications installed on the device. Kiosk Browser now has wake lock functionality to keep the screen on and can also be made to reload the start page if the network restarts.

Advantages:

- Has simple and minimalistic design
- Runs in full screen without navigation and notification bars
- Has hidden settings access
- Supports HTML5, CSS3, Javascript

Disadvantages:

- Settings of the launcher are lackluster
- Does not run on boot and cannot replace stock launcher
- Application is not stable on the newer versions of Android

4 CHALLENGES

In this section, challenges and problems that were encountered during the development of the project are listed along with their step-by-step solutions and some of the compromises that had to be done due to the time limitations or technical reasons.

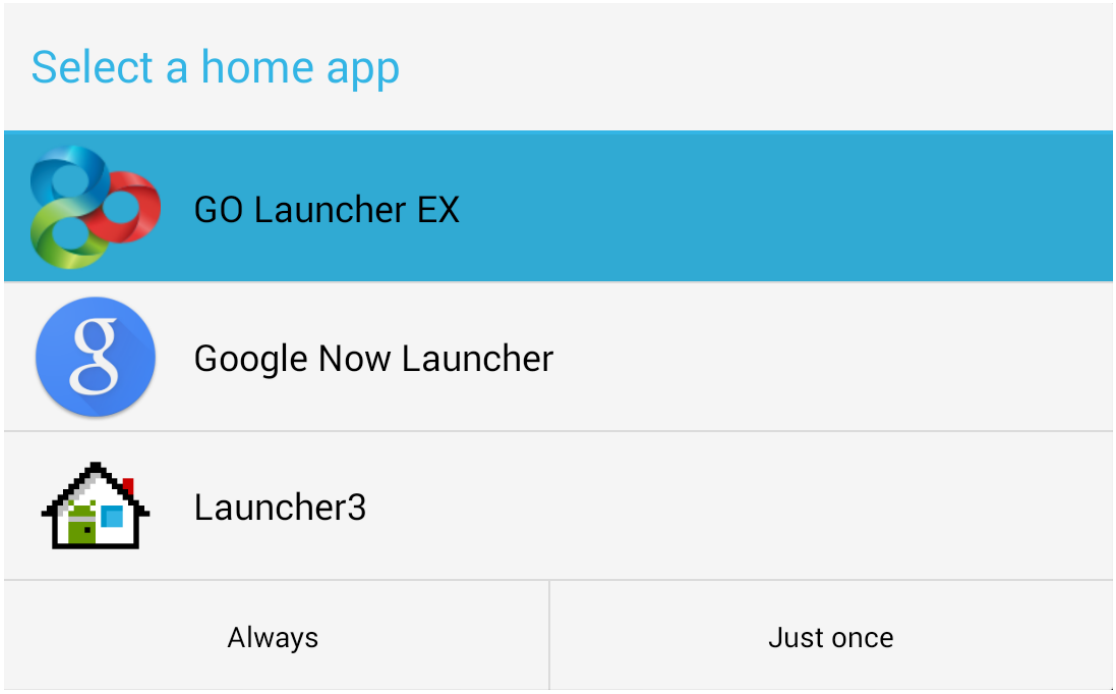
- Application as a launcher

To make an application launcher, only two lines of code should be added to the AndroidManifest.xml file in the main activity's intent-filter section. The code is as follows:

```
…
<activity
            android:name=" com.example.Example.MainActivity "
            …
                >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <!-- Next two lines are key to make application launcher-->
            <category android:name="android.intent.category.HOME" />
            <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
</activity>
…
```

- Be able to replace the stock launcher

That will be automatic, after installation of the app, pressing the Home button will bring up a dialog box similar to the one in Picture 1 where the desired launcher can be selected as a default for future use.



Picture 4. Launcher selection screen

- Run in full screen without any system bars.

This is a tricky requirement to fulfil since navigation and notification bars are running on system level and there is no option to permanently disable them on non-rooted phones or tablets. Therefore, to do that, the device has to be rooted first and then we can edit the build.prop file (located in *system* folder of every Android device), which contains many different settings for Android system.

> "Each Android build has a *build.prop* file . As the name suggests , it has important properties of that Android build. These properties are dependent on each device. Each build.prop file consists of numerous properties set by the Android build, each line for each property. "(Niranjan Thilak 2012)

We are going to take advantage of one of those settings for our purposes. The line `qemu.hw.mainkeys = 1` has to be edited or added to the end of the file.

- Have hidden settings access with at least two access levels

For this purpose, a transparent button was set up which calls the EditText dialog box. For security reasons, the button has a 5-second delay and the dialog has a timer to hide after several seconds of inactivity. Passwords for different access

levels can be set up in launcher settings, which will be discussed later.

In the program, the timer is used by creating a child of the timer class:

```
class MyTimerTask extends TimerTask {
                @Override
                public void run() {
                        handler.post(new Runnable() {
                                public void run() {
                                        dialog.dismiss();
                                }
                        });
                }
        };
```

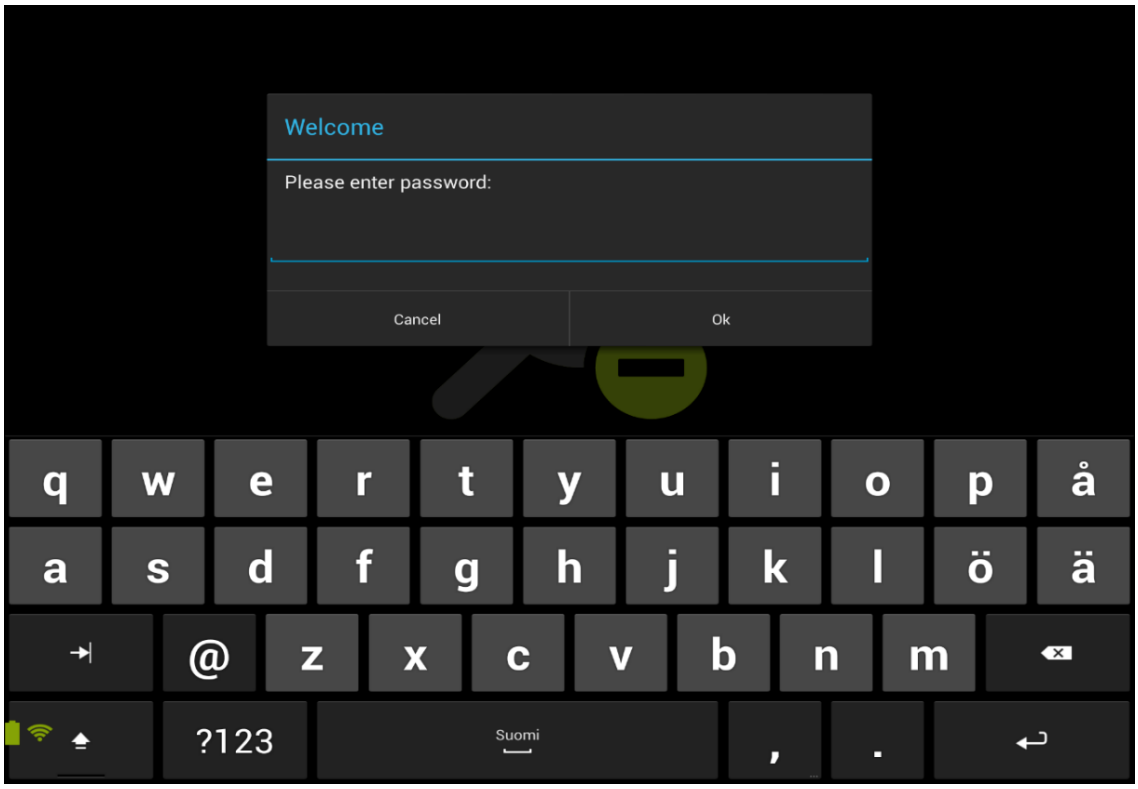
Starting the timer on button long-click:

```
timer = new Timer();
timertask = new MyTimerTask();
timer.schedule(timertask, seconds*1000L);
```

And restarting it on every text change action so that dialog does not dismiss it during user input, only in idle situations:

```
input.addTextChangedListener(new TextWatcher() {
        @Override
        public void onTextChanged() {
        if(timertask!=null){
                timertask.cancel();
                timertask = new MyTimerTask();
                timer.schedule(timertask, seconds*1000L);
        }
}
```

The result looks like this:



Picture 5. Enter password prompt

- Support HTML5, CSS3, Javascript, and possibly video and audio playback

HTML5, CSS3, and audio playback are supported by Android SDK's WebView by default and no further configuration needed for those technologies to work properly.

As for JavaScript, its support is available and can be enabled by the following line:

```
webView.getSettings().setJavaScriptEnabled(true);
```

Sadly, the Android SDK used in this project does not support HTML5 video playback natively – support for it was added later on in Android 4.4. However, there are ways to make the <video> tag work in the current WebView version, but it was decided that this support could be left for future improvements and was not required at the time.

- Support offline usage, i.e., store content locally until a connection is available

Support for offline usage and local storage was very high in the priority list, since this is an important point for device reliability and stability.

There are several ways in which fulfilling this requirement will benefit the experience with the device.

First of all, support for offline usage makes it possible for the device to be used outside network coverage and be able to collect and store answers. Those collected answers should be pushed to the server as soon as the device finds network next time. This is important for stationary devices in case of non-stable Wi-Fi coverage in the area where it is located as well as portable survey devices, so that they can be used on the go not depending on Wi-Fi or cell coverage and only come online once in a while to push collected answers to the server.

Secondly, local storage support makes devices much more reliable since this technology allows for the answers to be stored on device permanently even if it reboots or breaks. This is very important for reliability – even if something fails in the device, the answers will still be restorable and collectable from it in almost any case.

Local storage is used in place of another option such as cookies for several reasons. Cookies can be used to save persistent data on the device, but they have three disadvantages:

- Every HTTP request includes cookies, which slows web applications by needlessly transmitting the same information again and again.
- Since every HTTP request includes cookies, data is sent unencrypted over the internet (unless the whole website is served over SSL)
- Cookies are limited in size, only around 4 KB of information, which is not useful enough when there is an HTML5 local storage alternative available.

HTML5 storage answers all of the requirements for this project's usage:

- at least 5MB of storage, which is enough for months of stored answers without ever clearing them
- persists beyond device reboot or page refresh
- is transmitted to the server only when we need it

To fulfill the offline usage requirement, it was important to ensure that, whenever the device administrator navigates to the device settings and then goes back to the main activity with survey website loaded, this website and activity are not reloaded but are brought to front from the memory. This might look like a trivial task, but what makes it difficult is that programmer does not have power over Android's garbage collector and memory allocation. Therefore, in this example, whenever administrator navigates to device settings from the main activity, the garbage collector can destroy this main activity making our website to reload when we navigate back to it.

First, we have to make sure that device is not set up to destroy all activities as soon as they lose focus. The `Settings>System>Developer options>Apps>Don't keep activities` checkbox should be unchecked, because as subtitle to the setting states Android will "destroy every activity as soon as the user leaves it". This setting was on by default on the devices in the project and it was disabled to have offline usage experience.

Secondly, to make sure our activity is not collected and destroyed, our application should use reasonable amount of the device's memory. The current generation of devices has 2GB of RAM installed. The RAM usage of the application was monitored and it was found that amount of used memory floats around 50 megabytes as one can see from the table on the following picture:

```
C:\Users\Artur>adb shell dumpsys meminfo com.example.[redacted]
Applications Memory Usage (kB):
Uptime: 425990 Realtime: 425983

** MEMINFO in pid 2554 [com.example.[redacted]] **
                   Shared  Private     Heap     Heap     Heap
                     Pss    Dirty    Dirty     Size    Alloc     Free
                  ------  ------   ------   ------   ------   ------
      Native          0       0        0    14888     9308      167
      Dalvik      19096   12052    18328    25836    24834     1002
      Cursor          0       0        0
      Ashmem       1516       0     1516
    Other dev     11323    1420     1212
     .so mmap      7664    1760     2280
    .jar mmap         0       0        0
    .apk mmap       128       0        0
    .ttf mmap       642       0        0
    .dex mmap      1352       0        0
   Other mmap       373       8       24
      Unknown      9994     752     9952
        TOTAL     52088   15992    33312    40724    34142     1169
```

Picture 6. Application memory usage

This makes us believe that application is safe from being garbage collected when we navigate to other screens and practical tests supported this claim – website was not reloaded by Android even once when bringing the main activity back from memory.

Next we will explain how local storage was set up on the device itself. The server side of the requirement and code needed there will not be explained. However, it needs to be stated that for local storage to work properly it has to be set up the way that website saves answers to it. The client side and functions that allow website store answers on the device are shown in Figureor Table X. The following code gives website 5 MB of space for its needs to save text, files and other required information to store user feedback:

```
webView.getSettings().setDatabaseEnabled(true);
webView.getSettings().setDomStorageEnabled(true);
String databasePath = this.getApplicationContext().getDir("database",
Context.MODE_PRIVATE).getPath();
webView.getSettings().setDatabasePath(databasePath);
webView.setWebChromeClient(new WebChromeClient() {
public void onExceededDatabaseQuota(String url, String
databaseIdentifier, long currentQuota, long estimatedSize, long
totalUsedQuota, WebStorage.QuotaUpdater quotaUpdater) {
        quotaUpdater.updateQuota(5 * 1024 * 1024);
        }
});
```

- Have all settings in clear human readable xml file

There are several options to save persistent data in applications on Android. They include:

- Shared Preferences

For storing private primitive data in key-value pairs.

- Network Connection

For storing data on the web with your own network server.

- SQLite Databases

For storing structured data in a private database.

- Internal Storage

For storing private data on the device memory.

- External Storage

For storing public data on the shared external storage.

Shared Preferences were used in this project since they do not require constant Internet availability to save and retrieve data, like the Network Connection option. In addition, private application data can be stored, unlike with External Storage, and finally Android's software development kit provides easy access to the stored data and power of databases or direct file storage is excessive for the Kiosk Launcher's needs.

Using Shared Preferences is a common practice for storing configuration data in applications on Android because they allow storing of several types of information: booleans, floats, integers, longs, and strings. This data will persist across user sessions.

The Child from PrefernceActivity class loads the settings screen from xml file and Android will take care of saving settings for us. This is how settings activity loads settings:

```
public class SettingsActivity extends PreferenceActivity{
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
addPreferencesFromResource(R.xml.preferences);
...
}
```

The Xml file that was used in the previous code:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
xmlns:android="http://schemas.android.com/apk/res/android" >
    <PreferenceCategory android:title="Simple Preferences">
    <CheckBoxPreference
        android:key="pref_fullscreen"
        android:title="@string/pref_fullscreen"
        android:summary="@string/pref_fullscreen_summ"
        android:defaultValue="true" />
    <EditTextPreference
        android:key="pref_password_launcher"
        android:title="@string/pref_password_launcher"
        android:summary="@string/pref_password_launcher_summ"
        android:defaultValue="@string/def_password_launcher" />
        <EditTextPreference
            android:key="pref_password_wifi"
            android:title="@string/pref_password_wifi"
            android:summary="@string/pref_password_wifi_summ"
            android:defaultValue="@string/def_password_wifi" />
        <EditTextPreference
            android:key="pref_password_settings"
            android:title="@string/pref_password_settings"
            android:summary="@string/pref_password_settings_summ"
            android:defaultValue="@string/def_password_settings" />
    </PreferenceCategory>
</PreferenceScreen>
```

The Settings file will be stored automatically in the folder `/data/data/<application name>/shared_prefs/` in human readable form.

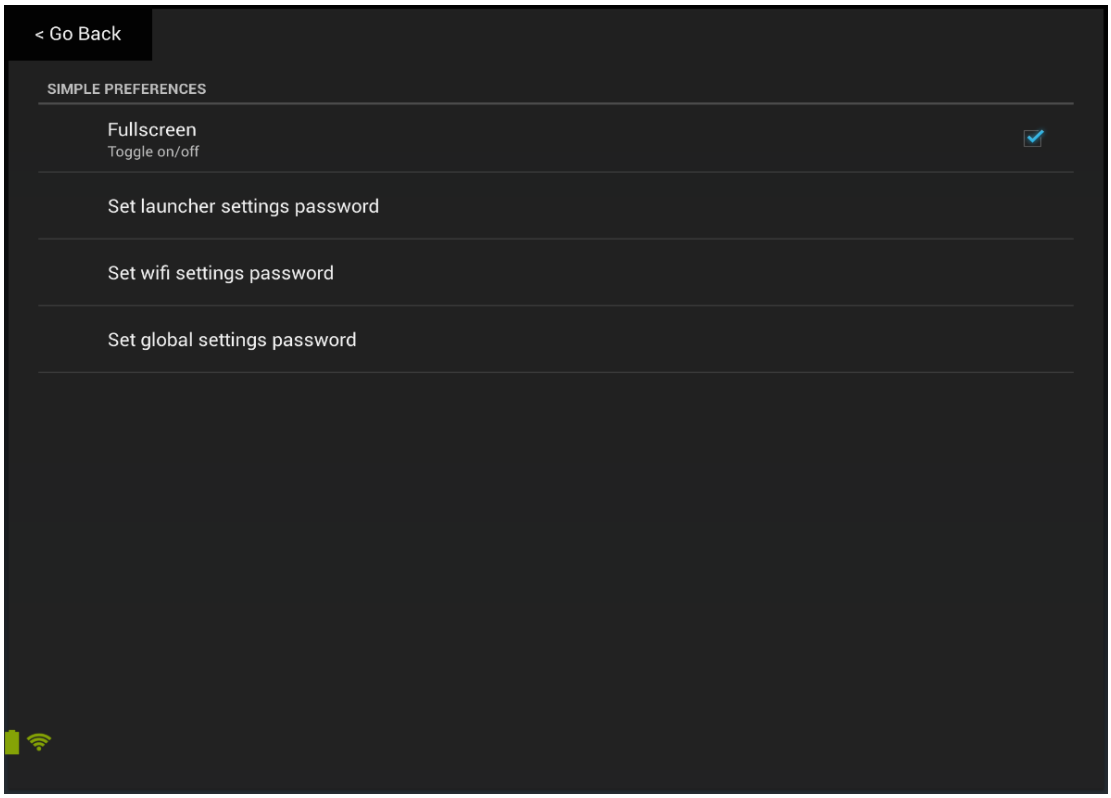
We access settings whenever we need in the application by creating SharedPreferences class object and instantiating it by calling getDefaultSharedPreferences (argument) function of PreferenceManager class and sending application context as an argument. Overall, it will look like that:

```
SharedPreferences prefs =
PreferenceManager.getDefaultSharedPreferences(getBaseContext());
```

The values are accessed individually by calling getBoolean, getStrings and other respective classes:

```
String example = prefs.getString("pref_password_example", "default");
```

Simple example settings screen:



Picture 7. Launcher settings screen

- Show battery and Wi-Fi statuses

This requirement is also one of the most important ones and it was the biggest challenge to implement such functionality.

Usually in the Android system, the battery status and Wi-Fi signal strength are pictured by icons in the notification bar. This project expects the web browser to run in full screen mode without any systems bars, therefore a different place had to be found for this purpose.

Icons are provided by the designer and reflect the company's style in color and minimalistic design:

The service application component was chosen to run and show status in the foreground. There are several reasons for that.

Firstly, services are designed to execute in the background and perform long-running operations, which fits our purpose. Secondly, they do not provide a user interface and that makes them lightweight and fast performing. Additionally, the service will run indefinitely even if the activity stays in the background until another component stops it or it terminates itself. Finally, services have high priority in the system, higher than the background or hidden activities.

There are two types of services – started and bound. Bound services provide a client-server interface between itself and the activity that started it, but the client-server interface can only run as long as the bound application component is alive. Because of this limitation, the “started” service was chosen as a basis for Wi-Fi and battery status icons.

Battery service responds to the universal Android intent `ACTION_BATTERY_CHANGED` which has information on the charging state, battery level, its health and other extra data. The main function of the battery service gets figures from the intent and updates internal values to match them:

```
private void calculateBatteryLevel(Intent intent){
if (Intent.ACTION_BATTERY_CHANGED.equals(intent.getAction())){
    int level = intent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
    int scale = intent.getIntExtra(BatteryManager.EXTRA_SCALE, 100);
    int batteryLevelPercent = level * 100 / scale;
    int status = intent.getIntExtra(BatteryManager.EXTRA_STATUS,
BatteryManager.BATTERY_STATUS_UNKNOWN);
if (m_batteryLevelPercent != batteryLevelPercent || m_batteryStatus !=
status){
    m_batteryLevelPercent = batteryLevelPercent;
    m_batteryStatus = status;
    updateBattery();
    }
  }
}
```

Then image views are updated properly according to the readings from the intent:

```
public void updateBattery(){
if(m_batteryLevelPercent>=90)
        imageview.setImageResource(R.drawable.battery100);
else if(m_batteryLevelPercent>=70)
        imageview.setImageResource(R.drawable.battery80);
```

```
else if(m_batteryLevelPercent>=50)
        imageview.setImageResource(R.drawable.battery60);
else if(m_batteryLevelPercent>=30)
        imageview.setImageResource(R.drawable.battery40);
else if(m_batteryLevelPercent>=10)
        imageview.setImageResource(R.drawable.battery20);
else
        imageview.setImageResource(R.drawable.battery20);
if(m_batteryStatus==2)
        imageviewStatus.setImageResource(R.drawable.charching);
else
        imageviewStatus.setImageResource(android.R.color.transparent);
}
```

The logical scheme is represented as follows:

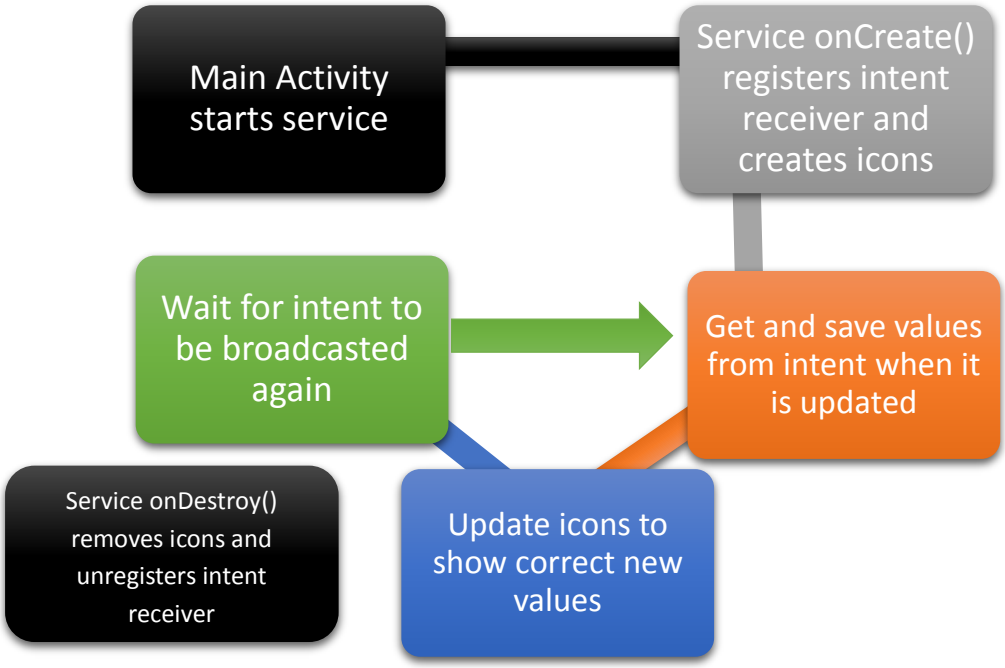


Figure 3. Battery service logical scheme

The Wi-Fi service is based on battery service but has a slightly more complex structure because Android does not provide a built-in function to check connectivity to the internet. One of the ways to implement this functionality was to open http connection to the server and get the response code from the server. Launcher checks for three levels of internet connectivity:

- Wi-Fi network connectivity

The Launcher checks if a local Wi-Fi network exists and if the device can connect to it and get a proper IP address.

- Connectivity to the Internet

  The Launcher checks if connectivity to general Internet areas exists by getting the response code from the Google website as one of the most reliable pages.

- Connectivity to the company's server

  The Launcher gets the response code from the server with a questionnaire to verify that the device can download and update its survey.

The first check was achieved with the following code:

```
…
cm = (ConnectivityManager)this.getSystemService
(Context.CONNECTIVITY_SERVICE);
if (cm.getNetworkInfo(1).getState() == NetworkInfo.State.CONNECTED){
…
```

The second and third checks were merged into one function:

```
private class CheckConnectivity extends AsyncTask<URL, Void, Integer>{
        int result1 = 0;
        int result2 = 0;
        protected Integer doInBackground(URL... urls) {try{
        HttpURLConnection urlc = (HttpURLConnection)
(urls[0].openConnection());
        urlc.setConnectTimeout(3000); //choose your own timeframe
        urlc.connect();
        result1 = urlc.getResponseCode();
        HttpURLConnection urlc1 = (HttpURLConnection)
(urls[1].openConnection());
        urlc1.setConnectTimeout(3000); //choose your own timeframe
        urlc1.connect();
        result2 = urlc1.getResponseCode();}}
```

In the same CheckConnectivity task we set the status icon to reflect the results that we received from our checks:

```
protected void onPostExecute(Integer result) {
        if(result2==200)
```

```
        imageviewStatus.setImageResource(R.drawable.vpn);
        else if(result1==200)
        imageviewStatus.setImageResource(R.drawable.internet);
        else
imageviewStatus.setImageResource(android.R.color.transparent);
}
```

In case website is reachable, it responds with the result code “200”. Depending on which website answered with this code, we either indicate to the user that we have connection to the company’s server, Internet or neither.

5 FUTURE POSSIBILITIES

There are several ways to improve this project. Some of the features were not implemented due to a limited timeframe and others were thought of and asked by the commissioning company after the initial requirements list was set. This project will continue to evolve and worked on and any unexpected interactions will be fixed, the application response time will be increased, as well as stability and new requirements that came up and might come up in the future will be met; some of the new requirements are listed below.

1. Video and audio playback

Although this is not the most requested option, but nonetheless it is handy to have the ability to show a commercial video or screensaver to attract people to answer the questionnaire on the device.

The author has experience with implementing this functionality in conjunction with Arduino device connected to motion detector to play video whenever someone walks by the tablet computer, however it was not added to this launcher at the time of the writing of this thesis.

There are several considerations to note while developing this role for the application.

First of all, video to be played should be in one of the natively supported Android formats. This list is quite limited yet, but expanding with new firmware releases:

Table 1. Supported video formats

Format / Codec	Encoder	Decoder	Details	Supported File Type(s) / Container Formats
H.263	•	•		• 3GPP (.3gp) • MPEG-4 (.mp4)
H.264 AVC	• (Android 3.0+)	•	Baseline Profile (BP)	• 3GPP (.3gp) • MPEG-4 (.mp4) • MPEG-TS (.ts, AAC audio only, not seekable, Android 3.0+)
MPEG-4 SP		•		3GPP (.3gp)
VP8	• (Android 4.3+)	• (Android 2.3.3+)	Streamable only in Android 4.0 and above	• WebM (.webm) • Matroska (.mkv, Android 4.0+)

Secondly, one should keep in mind the MediaPlayer class state machine:

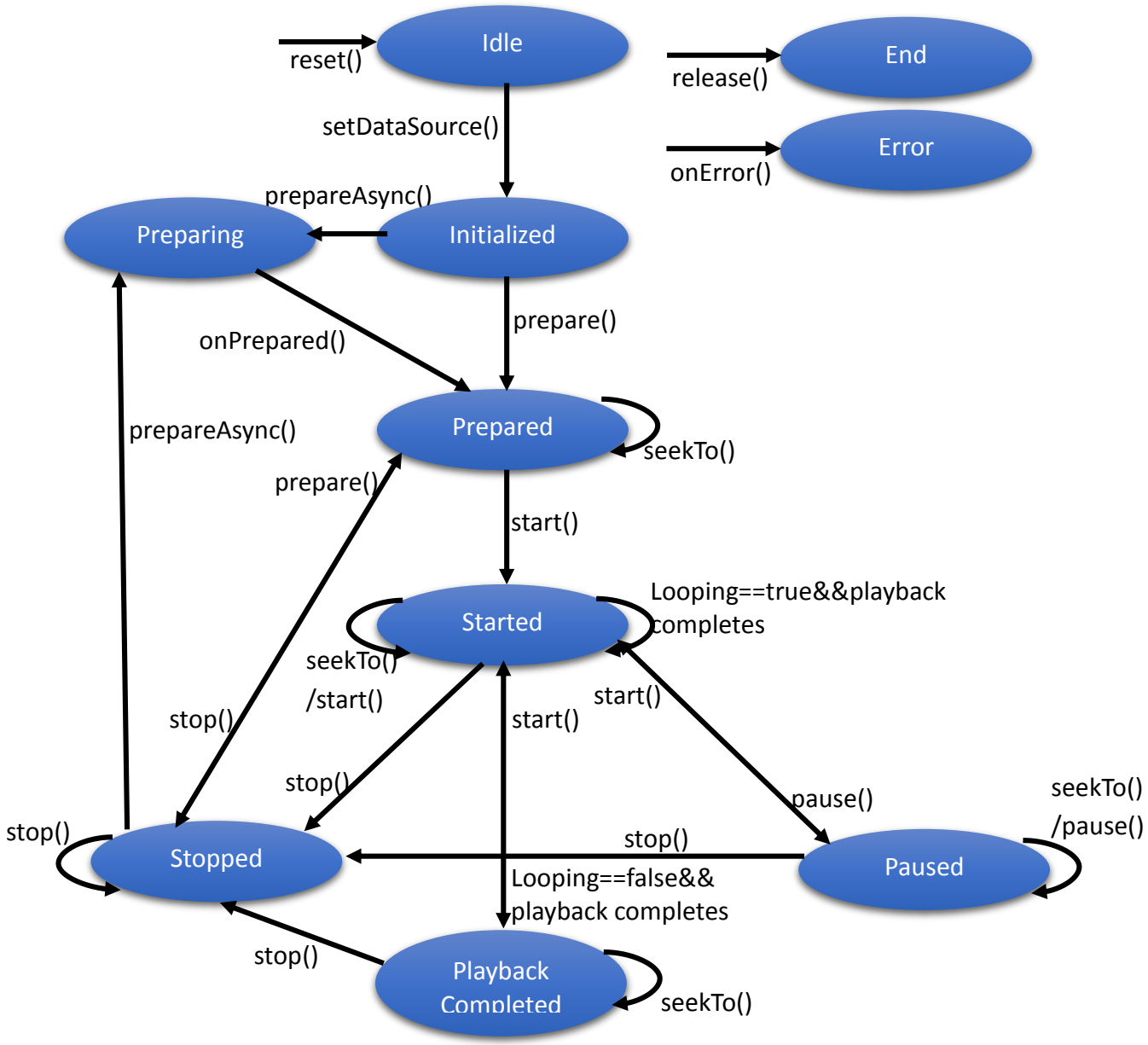


Figure 3. MediaPlayer state machine (Google developers website 2015)

Using this, it is planned to implement video and audio playback for the launcher as one of the upgrades later on.

2. Orientation change

The Launcher now exists only in horizontal mode and the device can be turned 180 degrees to better fit in the case without negative consequences. However, vertical orientation can be added in the future to support the upright mode which

fits better some types of questions in the poll.
This feature is among one of the most requested ones from the customers and it is sure that it will come to life in the future updates although it was not yet added because of the redesign to the website that is needed for the vertical mode to fit and feel proper. When the upright layout is ready, the option to change orientation will be added to the settings menu.

By itself orientation on Android is simply changed by configuring `android:screenOrientation` property in the manifest file for the application and can take one of the following self-explanatory attributes: unspecified, behind, landscape, portrait, reverseLandscape, reversePortrait, sensorLandscape, sensorPortrait, userLandscape, userPortrait, sensor, fullSensor, nosensor, user, fullUser, or locked.

3. Tap-tap wakeup

Among the nice-to-have features that were requested after the launcher development had started, tap-tap wakeup is one of the most interesting. However, tap-tap wakeup is only applicable to portable questionnaire devices. It allows the tablet to go to sleep (slow its processing speed, turn off display etc.) when laying it down on the table to save battery power and bring it back to life by lifting it up or tapping in a predefined way on the screen.

During development of this element, frustrating problem was encountered that some models of Android tablets disable their gyroscope readings as soon as they go to sleep. Thus, the device gets a sensor reading that it is laid down on the table and is able to slow its processor and disable screen, but, at the same time, it deactivates the accelerometer and gyroscope and is unable to understand that it is lifted up and needs to wake up. One of these models was a primary target for the launcher so this feature had to be dropped.

On the other hand, it was possible to implement and test a small tap-tap wakeup application; devices can understand whenever they are laid on the table and then wakeup from a simple double tap on the screen. This program is now ready to be modified to work in the application and will be added in the coming updates.

6 CONCLUSION

Android is a rapidly growing operating system that has plenty of use on mobile phones, tablets, TV sets, and embedded devices. More and more ways to use those devices are found every day.

This thesis has demonstrated that among its many user cases Android can be deployed as a questionnaire platform to collect customer feedback and explained how one of the backend applications was built to support this use. Although the Android operating system still has some minor complications for the developers, Google provides new tools, tutorials and additional features with each release. No other mobile operating system can compete in price, user base coverage, or freedom for the developers nowadays.

We hope that this thesis will be of use to someone else who is starting Android application development in general and Kiosk Launcher development in particular. It was really enjoyable to work on this project and to continue to provide support and updates for the launcher.

REFERENCES

42Gears Mobility Systems, 2015. SureLock Kiosk Lockdown. [program] Google Play. Availbale at: <https://play.google.com/store/apps/details?id=com.gears42.surelock> [Accessed April 2015]

International Data Corporation, 2014. Smartphone OS Market Share. [online] Available at: <http://www.idc.com/prodserv/smartphone-os-market-share.jsp> [Accessed 15 April 2015]

Nickinson, P. 2012. Android A to Z: What is a launcher? Androidcentral blog. [blog] 24 January. Available at: <http://www.androidcentral.com/android-z-what-launcher> [Accessed 20 April 2015]

Powell, A. 2012. Kiosk Browser. [program] Google play. Available at: <https://play.google.com/store/apps/details?id=it.automated.android.browser.kiosk> [Accessed April 2015]

ProCo IT, 2015. Kiosk Browser Lockdown. [program] Google play. Available at: <https://play.google.com/store/apps/details?id=com.procoit.kioskbrowser> [Accessed April 2015]

Statista, 2015. Global market share held by the leading mobile operating systems from January 2012 to March 2015. [online] Available at: <http://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/> [Accessed 15 April 2015]

Thilak, N. 2012. Improving performance of your android device through optimizing build.prop settings. Niranjan Thilak blog. [blog] 12 June. Available at: <http://niranjanthilak.com/improving-performance-of-your-android-device-through-optimizing-build-prop-settings/> [Accessed 20 April 2015]